Lisp family,

有许多种Lisp：pure,common lisp,scheme….我们这里研究的是common lisp

lisp中基本元素：

atom(数字，字母)

list object/expression (a,b,c)

string:一系列char并且有双引号double quote

所对应的return value:

数字，自己就是自己，

字母，要加’符号，’a=A

String,”Hello”,return的就是”Hello”

T(true)

NIL(false或empty)

function:

14-2\*3在Lisp中表达



括号内第一个参数-被认作是function，我们需要在第一个参数写function的名字，后面的两个参数 14 （\* 2 3）被认作是 参数



arity:用来描述这个function所能采取的argument

只支持一个argument的function叫做 unary function (arity 1)

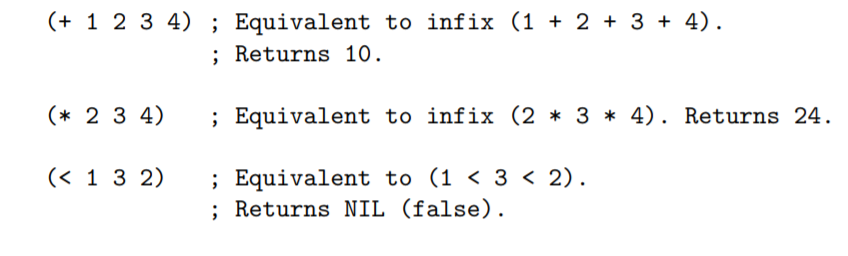
两个的叫做binary function (arity 2)

三个的叫做ternary function (arity 3)

n-ary function takes n arguments.

然后还有Variable arity function，这种可以支持任意个argument

比如运算符号和比较符号，我们并不在乎有几个argument

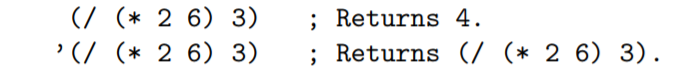


quiote能给字符return一个值





注意加了quote以后，他后面不加空格的一长串都认作为string



不会做具体运算

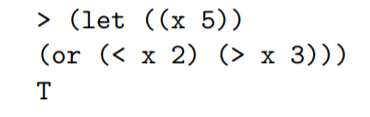
lisp的比较



Lisp的boolean logic :or and not



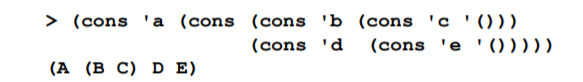
Lisp的赋值LET



三个用来创建List的build-in function



1.cons：将一个元素作为head加到一个现有List里去

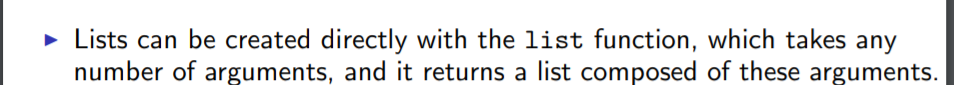


（cons a）:两个问题，a没加quote，二没有第二个LIST参数

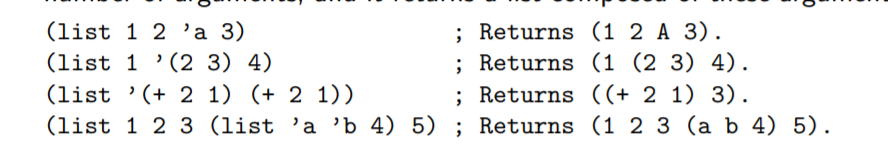
(cons ‘a ‘())答案是(A) 注意在空LIST前面也要加quote’

(cons ‘a) 依然不行，因为第二个List

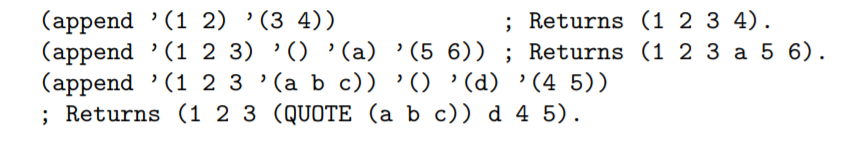
(cons（+ 2 3）’（b c）)=(A B C)

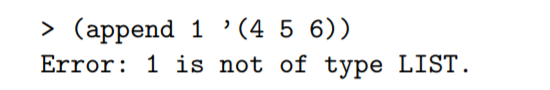
2.list 

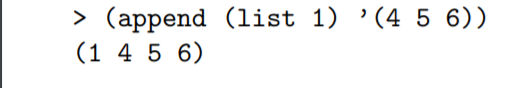
直接list 你所需要List的任意数量argument



3.append:把两个list连接起来，去除前后括号



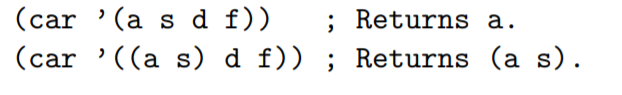


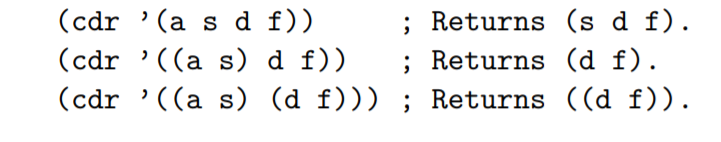


怎么access一个LIST

一共两种方法，car,cdr

car可以求出head，cdr可以求出tail





注意cdr求出的tail是list所以是有括号的

求出LIST中的第二个元素，先cdr求出tail,在car求出head



想求出第几个元素，就cdr n-1次，再car

（cdr(append ‘() ‘() ‘())）答案是NIL，因为empty list没有head or tail

NIL也可以只待empty

(cdr (list ‘() ‘(a))

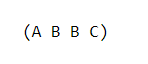
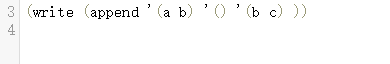
答案是(A)，因为list ‘() ‘(a)的答案是(NIL A),空集会作为nil显示

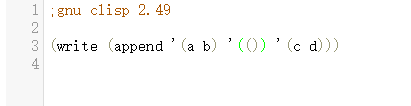
LIST 会把空集写成NIL



而append中如果直接出线空集

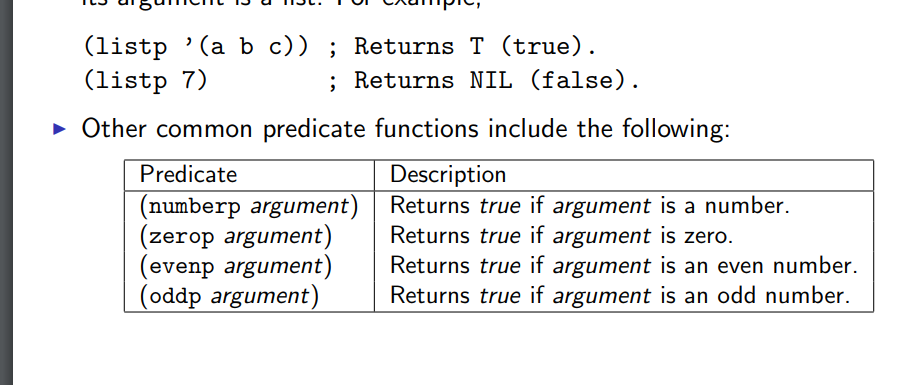
省略，因为空集前后括号都被去掉了

除非在空集中又加了一个空集，左右括号去掉就是空集



Predicate functions 谓词函数

作用是确认括号里的是不是数字、字母等等



Atom ,如果参数是atom就return true


 T

 T

|  |  |
| --- | --- |
| atom | 它接受一个参数，并返回t如果参数是一个原子或，否则nil。 |
| equal | 它有两个参数，并返回t，如果他们在结构上相同或否则nil |
| eq | 它有两个参数，并返回t，如果它们是相同的相同的对象，共享相同的内存位置或否则nil |
| eql | 它有两个参数，并返回t如果参数相等，或者如果他们是同一类型具有相同值的数字，或者如果他们是代表相同的字符的字符对象，否则返回nil |
| evenp | 它接受一个数字参数，并返回t如果参数为偶数或否则为nil。 |
| oddp | 它接受一个数字参数，并返回t如果参数为奇数或否则为nil。 |
| zerop | 它接受一个数字参数，并返回t如果参数是零或否则为nil。 |
| null | 它接受一个参数，并返回t，如果参数的计算结果为nil，否则返回nil。 |
| listp | 它接受一个参数，并返回t如果参数的计算结果为一个列表，否则返回nil。 |
| greaterp | 这需要一个或多个参数，并返回t，如果不是有一个单一的参数或参数是从左到右，或如果无先后，否则为nil。 |
| lessp | 这需要一个或多个参数，并返回t，如果不是有一个单一的参数或参数是从左到右依次更小的向右，或否则为nil. |
| numberp | 它接受一个参数，并返回t如果参数是一个数字，否则为nil。 |
| symbolp | 它接受一个参数，并返回t如果参数是一个符号，否则返回nil。 |
| integerp | 它接受一个参数，并返回t如果参数是一个整数，否则返回nil。 |
| rationalp | 它接受一个参数，并返回t如果参数是有理数，无论是比例或数量，否则返回nil>。 |
| floatp | 它接受一个参数，并返回t当参数则返回一个浮点数否则为nil。 |
| realp | 它接受一个参数，并返回t如果参数是一个实数，否则返回nil。 |
| complexp | 它接受一个参数，并返回t如果参数是一个复数，否则返回nil。 |
| characterp | 它接受一个参数，并返回t如果参数是一个字符，否则返回nil。 |
| stringp | 它接受一个参数，并返回t，如果参数是一个字符串对象，否则返回nil。 |
| arrayp | 它接受一个参数，并返回t如果参数是一个数组对象，否则返回nil。 |
| packagep | 它接受一个参数，并返回t，如果参数是一个包，否则返回nil。 |

Nullp ,如果参数是null， return true

（nullp NIL） T

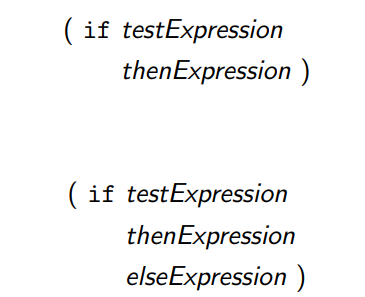
Equal,两个参数相等

 T

T

Interp ,参数是整数

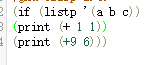
Lisp的control flow



TestExpression是判断句

THENEXPRESSION是表达

Else也是表达

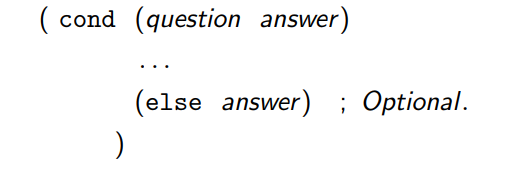






答案是一片空白

Multiple selection:cond



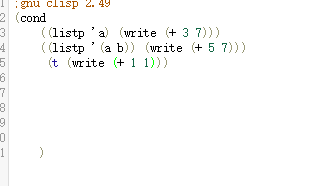
(cond

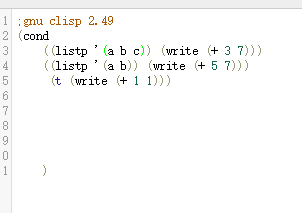
(test 1 action 1)

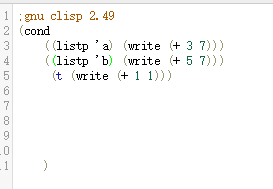
(test2 action2)

(testn actionn)

(t default action)

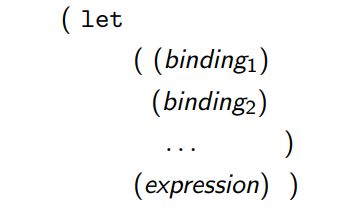
 12

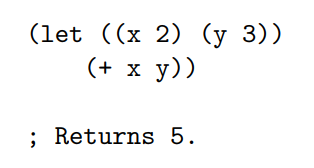
10

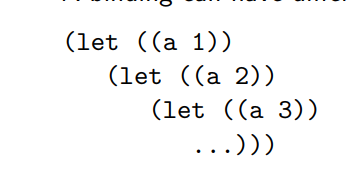
2  
listp 空集（）也是T

Binding :LET

Binding指的是对一个变量赋值

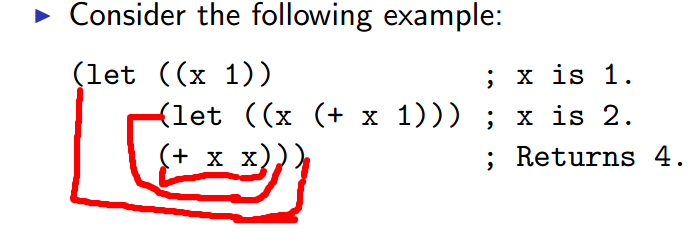




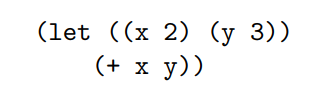


这时A的值是3

括号对应

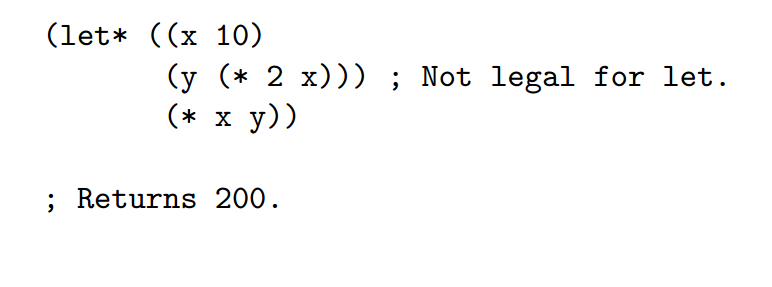


Let后面跟的是双括号为什么用双括号，最外面一个括号代表我这个let所要指定的值，里面的括号是给X值



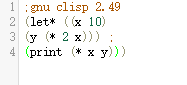
LET\*,当你需要赋值的VARIABLE建立在另一个VARIABLE上时

，Y的值建立在X上



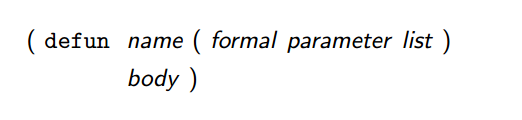
(y (\* 2 x))这里，如果是LET的话，X是没有值的，因为let只是赋值阶段，要到最后一步才会带入值

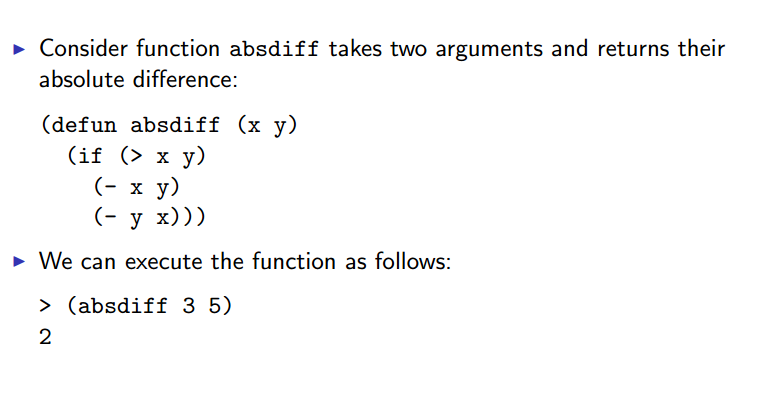
但是用Let\*就可以



描述一个function

用defun





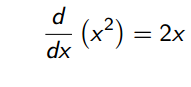
absdiff求两个参数之间差的绝对值

如果X大于Y，求X-Y不然就求Y-X

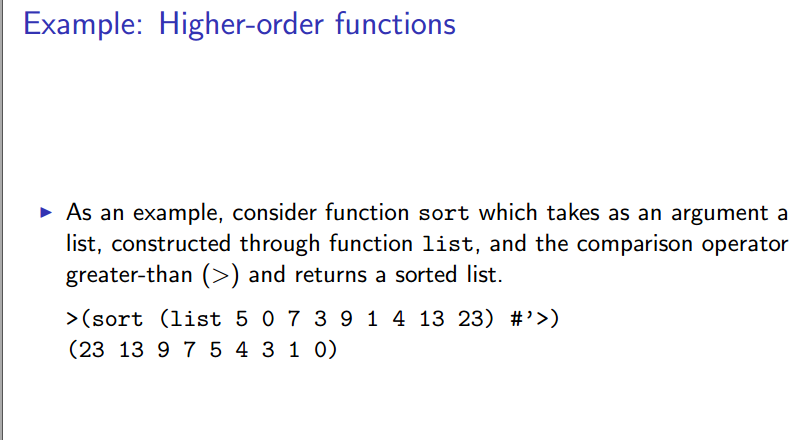
higher-order function

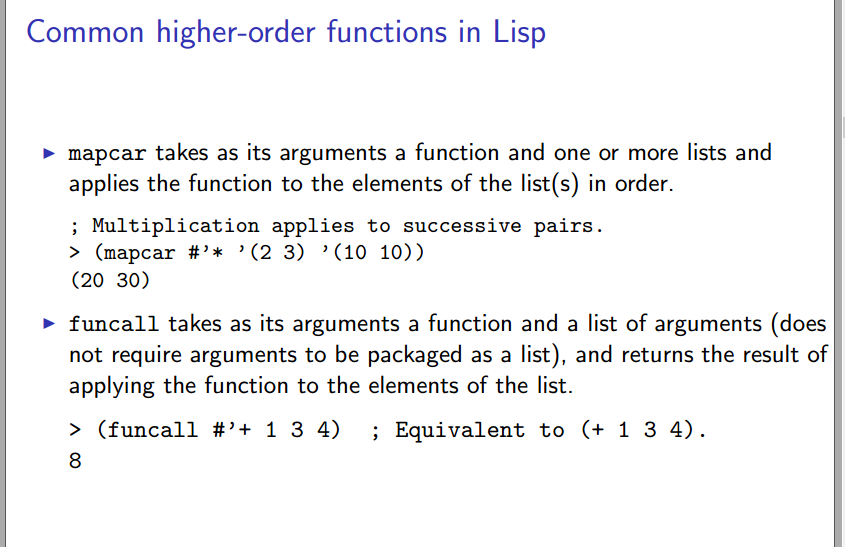
至少有其中一个特征：

1. 有一个或多个function作为参数
2. return 一个function

求导就是其中一种他让一个function转成了另一个function

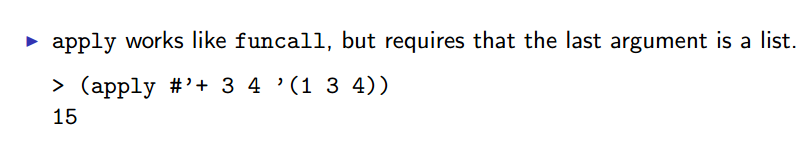
以内置function sort为例，他用一个List作为参数，通过function list来构造，然后又用return一个sorted list





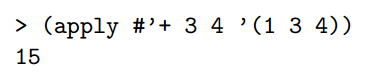
以mapcar为例，他的参数是一个function与一个或多个列表，按顺序将这个function应用到列表的element上

funcall 让一个function与一组argument作为argument，然后把这个function意义应用到这些element中//一组argument实际上是list，但是不需要加括号



apply与funcall相同，但是让最后一个argument是List

3+4+1+3+4=15



anonymous function 匿名function

一个匿名function是一个被定义的，有可能被call的但是没有绑定identifier(name)的function

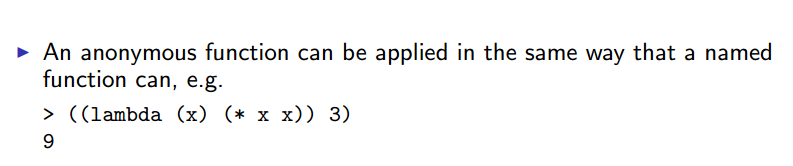
与defun定义的function不同。匿名function不存储在Memory中

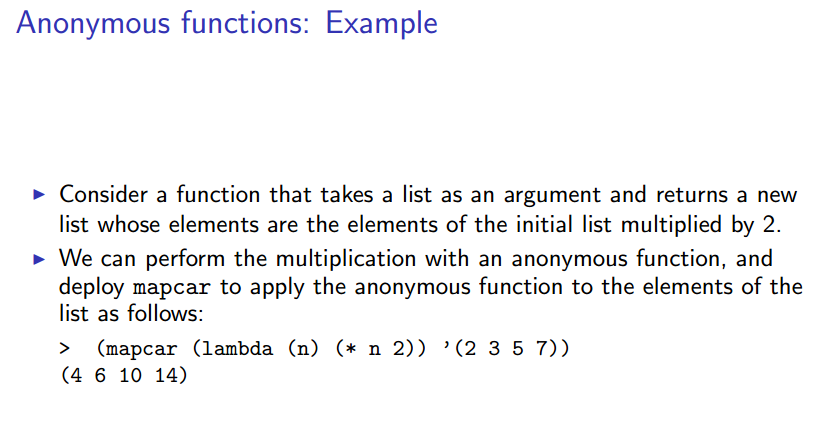
anonymous function也叫做lamda function

格式如下



example:





把一个list作为function，然后每个element x2

与mapcar合用，mapcar会把这个 function应用到每一个element上

variable与assignments

variable是global那么所有地方都能读取

local variable只能被{}block 内读取

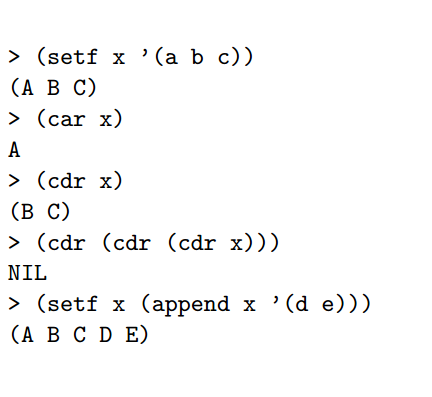
在block内，local variable优先读取，global value被稍后读取，如果都没有，就会产生error

setf 可以定义global 与local variable

setq只能定义global



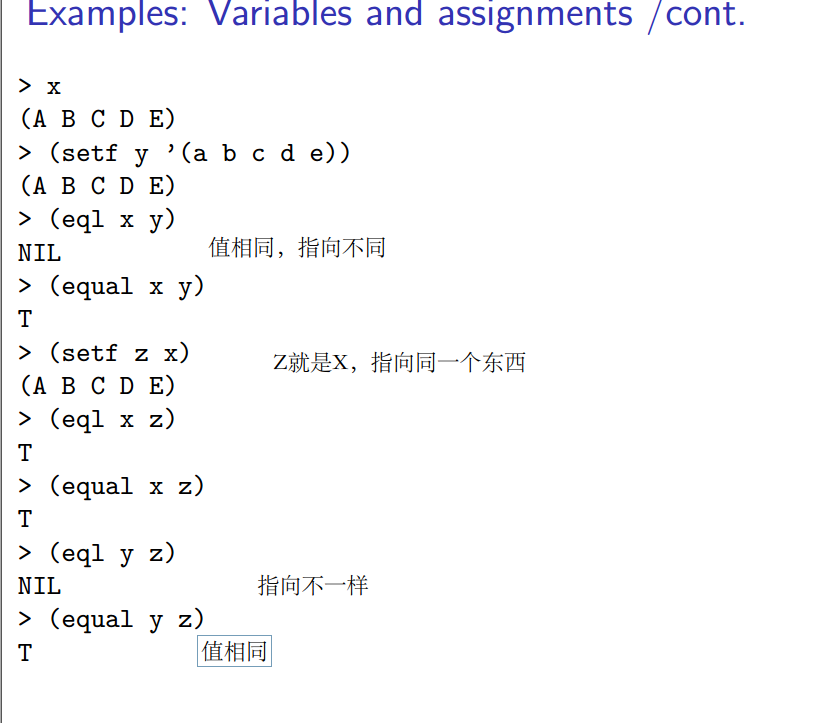
例子：



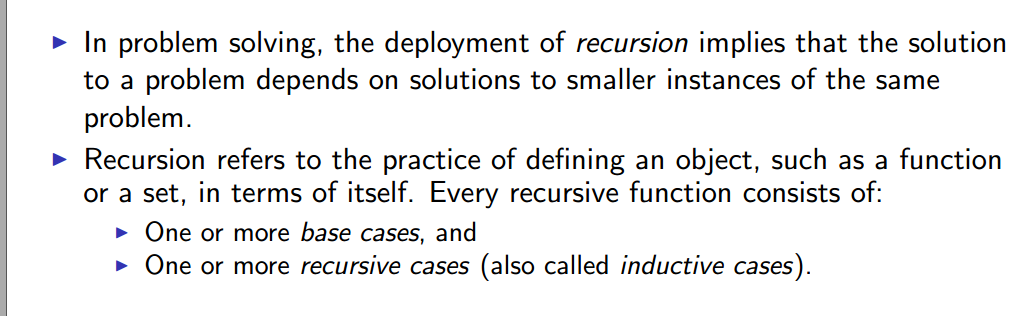
X是 ABC，global，注意了car cdr这个操作不会改变X本身，只有setf会改变

Variable本质上是指针

function eql会ereturn true如果他的参数指向同一个object，function equal会return true只要值相等就行



描述一个recursive function

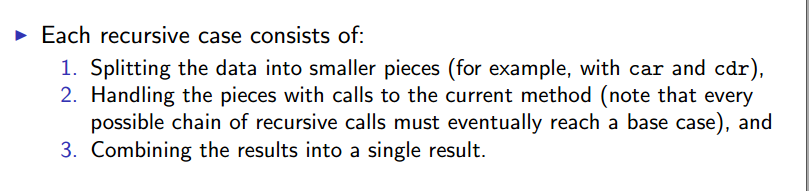


需要包含一个或多个base case

一个或多个recursive case （也叫做 inductive case）

在每个recursive case必须做到：

1. 把数据切分成更小块（例如car与cdr）
2. 用同样的method来handle这些小块
3. 把这些结果combine成一个单一结果

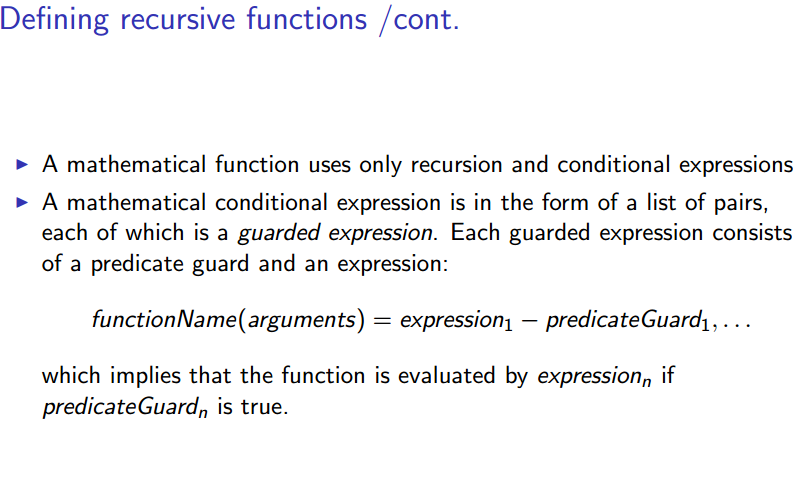


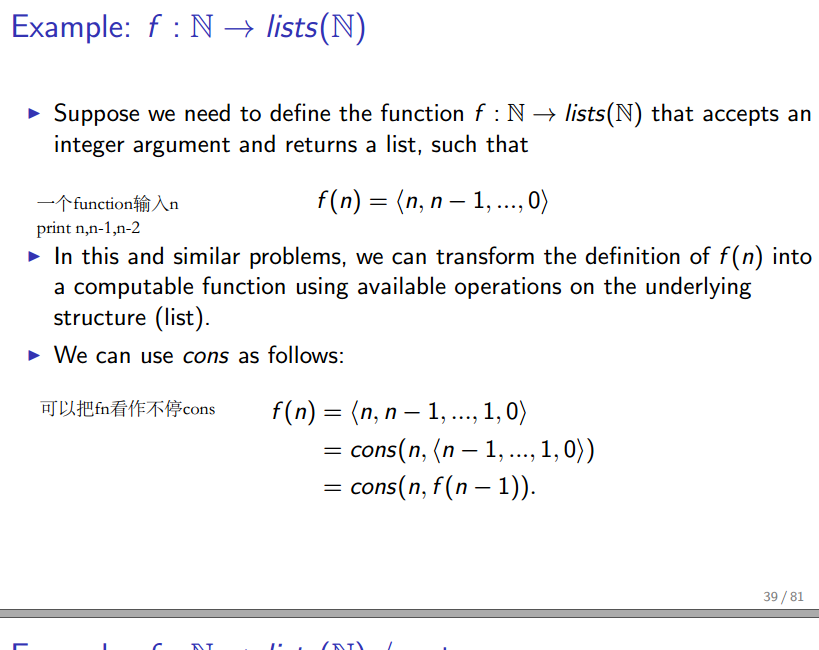
一个Mathematical function 只用recursion与conditional expression条件表达式

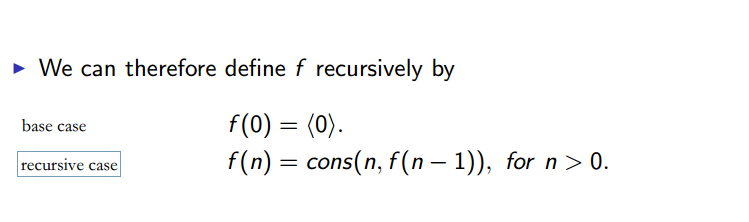
一个数学条件表达式是一组pairs，每一个pair都是一个guarded expression，每一个guarded expression包含了一个predicate guard 和一个expression

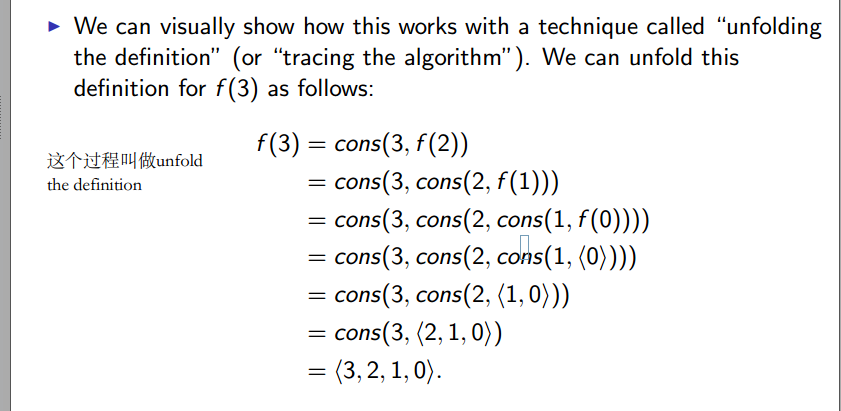


意思是function由expressionn执行如果predicateGuardn正确

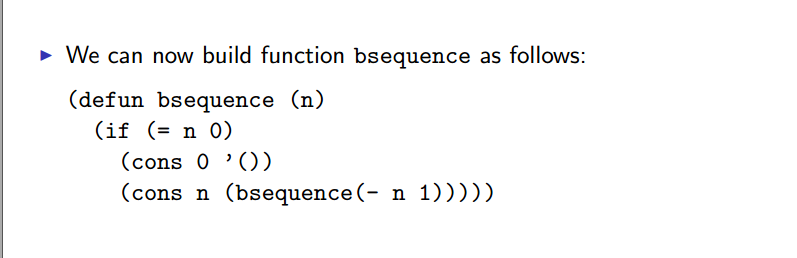




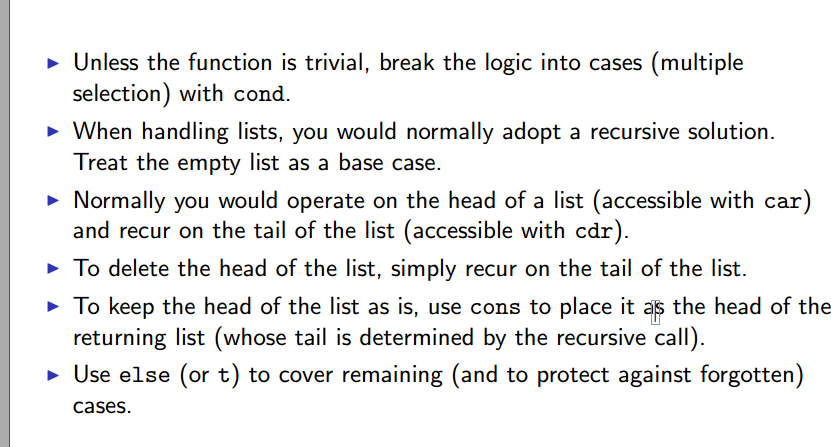




最终定义



很多例子都在pdf里



把逻辑break各种情况，用cond连接

如果是关于list，把空list作为base case，然后recursive的时候尽量让List变小

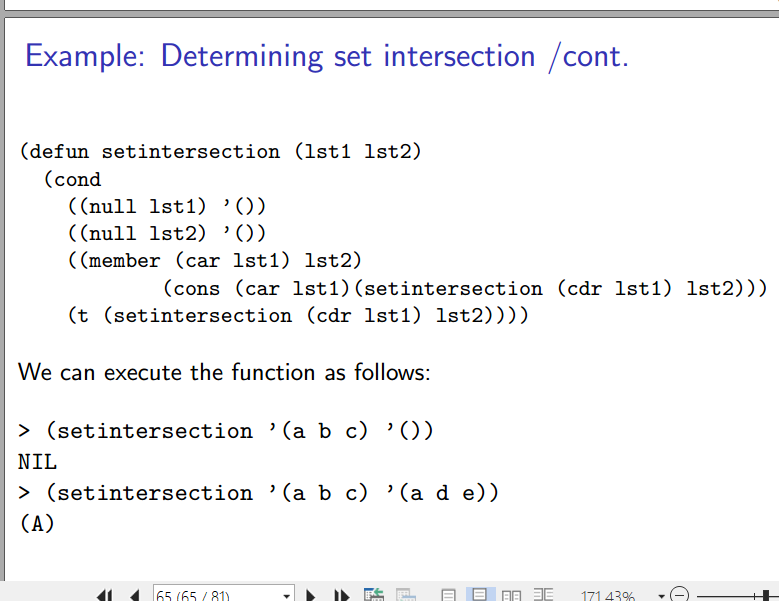
通常你需要对list 的head进行操作（car）,对list的tail进行recur(cdr)

为了删除list打的head，只用不停recur他的tail就行

为了保持head，用cons让他作为returnning list的head

用t来决定最后一个case

例子



base的都是空

不停地car，setintersection 剩下的

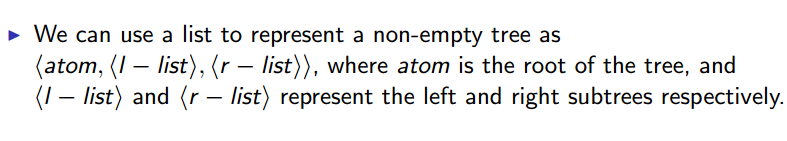
如果head是lst2中的一部分，留下，留下head的方法就是 cons，然后剩下的cdr recursive

不是lst2中一部分，舍去，舍去head的方法就是直接cdr recursive,

使用了member

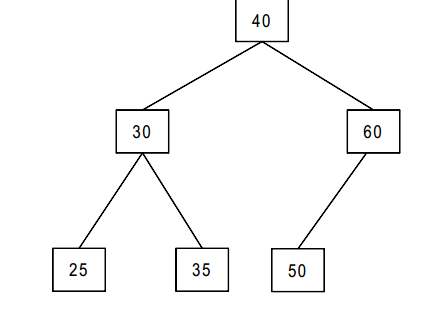
用来表达tree

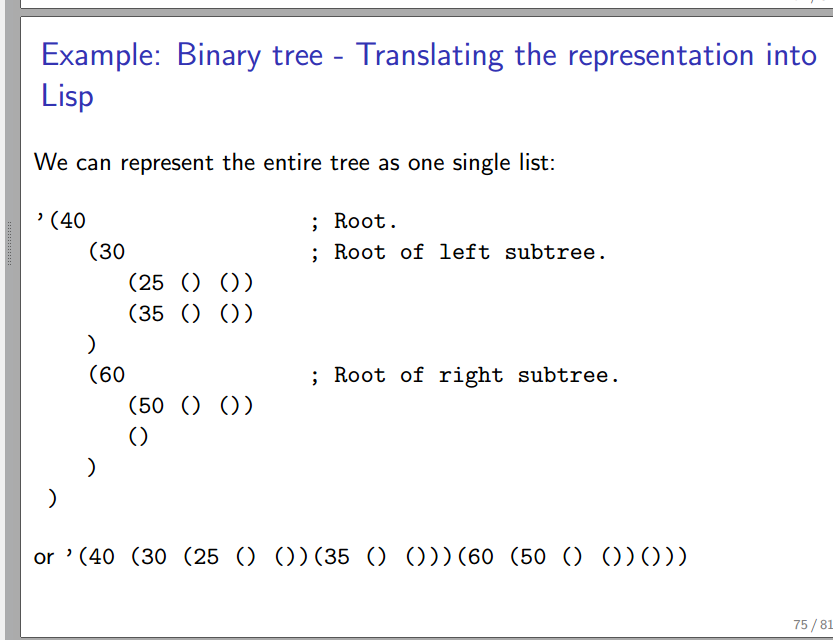
我们可以用List表达一个非空tree



atom表示root，左边右边分别表示左子树，右子树

例子





40是 root，分了两个大括号，30大括号是左子树，60大括号是右子树

怎么得到tree的元素

得到root使用car

bubble sort

